White Paper Report

Report ID: 98500

Application Number: HD5060109

Project Director: Natalia Smith (nsmith@email.unc.edu)

Institution: University of North Carolina, Chapel Hill

Reporting Period: 5/1/2009-4/30/2010

Report Due: 7/31/2011

Date Submitted: 8/7/2011

img2xml: Linking manuscript page images to TEI transcripts using SVG

Hugh A. Cayless, Co-Investigator (NYU)

Introduction

The *img2xml* project began with the observation that the Open Source tool *potrace* [1] could be used to generate Scaleable Vector Graphic (SVG) tracings of writing on a manuscript page. The goal of the project has been to start the development of a toolset for image-text linking and to explore the capabilities and limitations of the available tools. We very deliberately avoided attempting to construct an end-to-end solution for linking image and text, believing that a toolkit is a much more scaleable and extensible way to approach the problem. The main outcomes of this initial startup grant are a demonstrator project, *Verses and Fragments: The James L. Dusenbery Journal (1841–1842)* (http://docsouth.unc.edu/dusenbery/), and the source code hosted at https://github.com/hcayless/img2xml.

The tools and standards used in the course of this project include:

- Django (https://www.djangoproject.com/) – a web publishing framework written in the Python language
- Djatoka (http://djatoka.sourceforge.net/) – a Java-based open source image server
- The Gimp (http://www.gimp.org/) – an Open Source graphics editing tool
- ImageMagick (http://www.imagemagick.org/) – a software suite to create, edit, compose, or convert bitmap images
- Inkscape (http://inkscape.org/) – an Open Source vector graphics editor
- jQuery (http://jquery.com/) – a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.
- potrace (http://potrace.sourceforge.net/) – a utility for tracing bitmaps
- Python scripts for detecting lines of text (https://github.com/hcayless/img2xml)
- Scaleable Vector Graphics (SVG, http://www.w3.org/Graphics/SVG/) – an XML-based format for vector graphics
- The Text Encoding Iinitiative (TEI, http://www.tei-c.org) – an XML-based format for texts
- OpenLayers (http://openlayers.org/) – an Open Source javascript web mapping framework
- XSLT (http://www.w3.org/standards/xml/transformation) – a language for transforming XML documents

It should be noted that while the *Dusenbery Journal* made use of all of these in its development, it does not exploit them all in its web presentation. The goal was to produce a text side-by-side with a zoomable image, where lines of text were linked to lines in the page image. This could have been achieved with SVG, but OpenLayers provides similar functionality and supports browsers that lack SVG support.

The problem domain

An SVG tracing (and the SVG format in general) offers a means to deal effectively with some of the problems surrounding the linking of image to text. The Text Encoding Initiative (TEI) Guidelines provide mechanisms for linking image and text, but these suffer from the following issues:

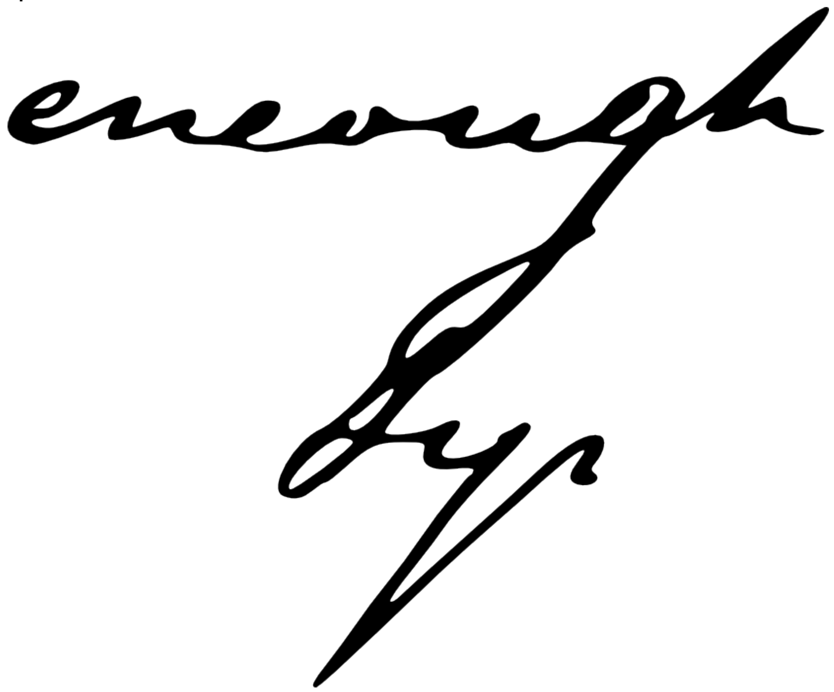
1. If one wishes to describe the relationship of, e.g. a line of text in a transcription to a zone in an image of the text, one must employ a coordinate system to describe the zone in the image. If the link is to a single image, the coordinate system employed may be based on the pixel size of the image, but this works less well when there are multiple versions (e.g. different resolutions or different shots) of the text image. TEI does this with the tei:facsimile element's children tei:surface, which defines the bounds of the text-bearing surface and tei:zone, which defines a rectangular or polygonal space within a tei:surface. The nature of the coordinate system tei:facsimile employs is underspecified, however. It does not specify units or orientation (though the examples given seem to assume pixels and a top-down orientation).
2. The limitations of TEI's implementation of facsimile coordinate systems mean that a generic solution, in which, for example, the given coordinates correspond to physical measurements rather than pixel dimensions would require extra definition that TEI isn't set up to handle. Such a solution requires the ability to perform coordinate system transformations.
3. The zone's shape must be defined. Until recently, TEI only permitted rectangular zones, identified by the x or y coordinates of their corners. The attribute @points on tei:zone has recently been added, permitting polygons to be represented.
4. TEI, because of its reliance on XML's tree structure, and the way in which its semantics are bound to that structure, may have trouble marking a line of text as being the unit corresponding to a zone in an image.

SVG offers at least a partial solution to problems 1, 2, and 3. It seamlessly handles multiple coordinate systems, scaling and translation, and supports a variety of shapes, including complex paths. Moreover, since it is an XML-based format, shapes may be given unique identifiers and pointed to using standard techniques. GIS tools, like OpenLayers, offer similar capabilities. This paper will outline the solution to #4 used in the Dusenbery project, which is on the simple, but kludgy end of the spectrum, and discuss some possibilities for more complete solutions.

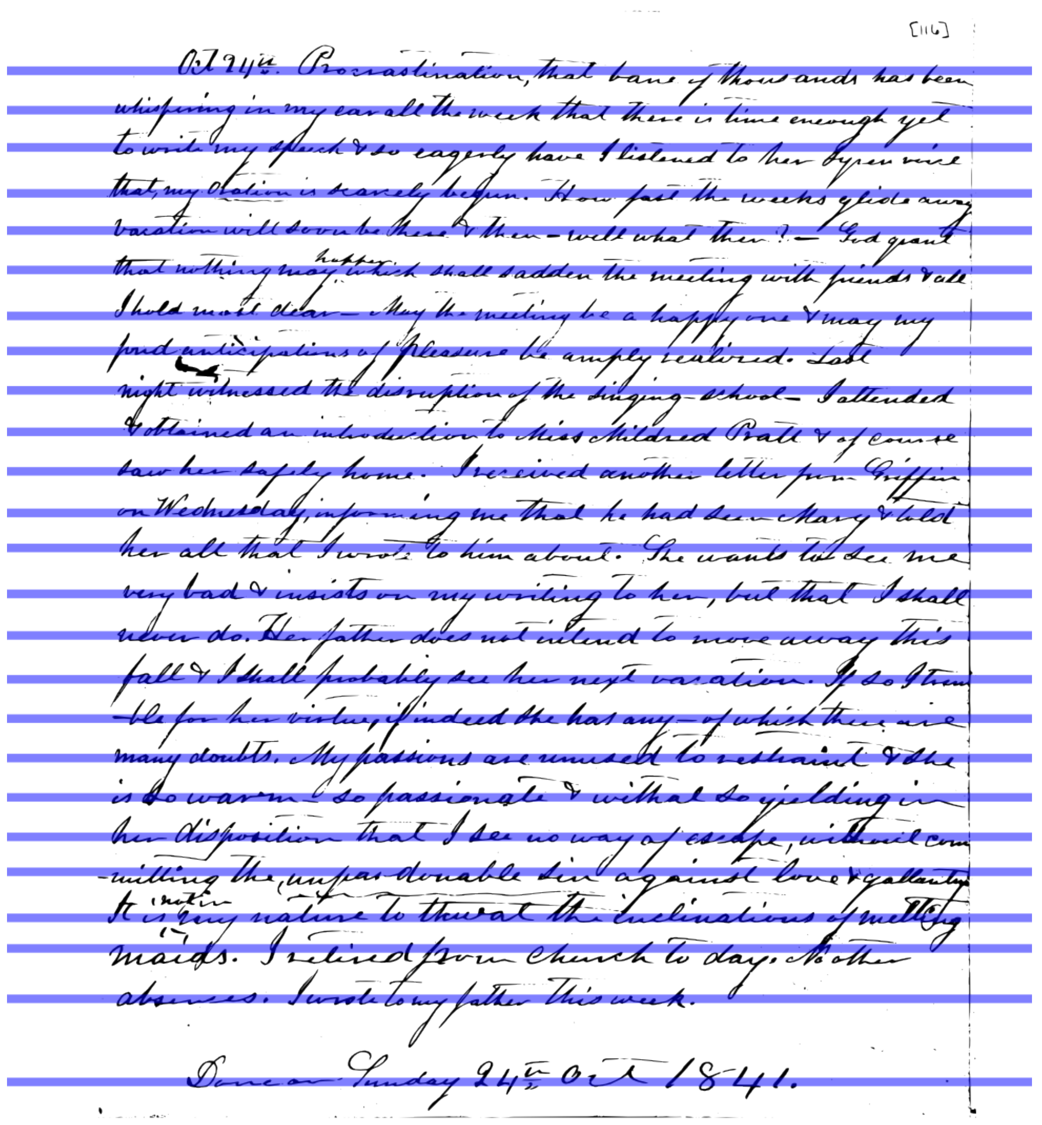
Demonstrator: James Dusenbery's Journal

http://docsouth.unc.edu/dusenbery/journal/

The journal is a handwritten document, authored by a student at the University of North Carolina at Chapel Hill in the 1840s. Roughly the first half of the journal consists of songs and poems Dusenbery liked, and copied, and the latter half begins as a conventional journal and concludes with copies of several letters (see http://docsouth.unc.edu/dusenbery/about/). It thus presents a nice mixture of verse and prose, and provided a good test for our line-recognition workflow. The SVG tracing converts contiguous segments of text into SVG paths, a path being a closed series of points connected by lines or Bézier curves, and forming a complex shape. Dusenbery's handwriting is cursive and fairly florid, with ascenders and descenders that frequently touch the line above or below (see e.g. lines 2-4 of http://docsouth.unc.edu/dusenbery/journal/#jld-p116). This means that strokes which touch across line boundaries will be a single svg:path, as can be seen in the example below:



Clearly, line detection is not simply a matter of finding SVG objects that line up horizontally. The line detection algorithm employed for the project works by finding peaks in the number of average-or-greater-sized svg:paths on the Y-axis of the document. These peaks tend to be located close to the baseline of a line of text. The program throws out peaks that are too close together to be separate lines. The algorithm is fairly naïve, but has proven effective at finding lines in the Dusenbery journal, and in tests on other material. The figure below illustrates the peaks detected by the `line_detector.py` program.


[116]

Oct 24th. Procrastination, that bane of thousands has been whispering in my ear all the week that there is time enough yet to write my speech & so eagerly have I listened to her Syren voice that, my Oration is scarcely begun. How fast the weeks glide away vacation will soon be here & then — well what then? — God grant that nothing may happen which shall sadden the meeting with friends & all I hold most dear — May the meeting be a happy one & may my fond anticipations of pleasure be amply realized. Last night witnessed the disruption of the singing-school — I attended & obtained an introduction to Miss Mildred Pratt & of course saw her safely home. I received another letter from Griffin on Wednesday, informing me that he had seen Mary & told her all that I wrote to him about. She wants to see me very bad & insists on my writing to her, but that I shall never do. Her father does not intend to move away this fall & I shall probably see her next vacation. If so I tremble for her virtue, if indeed she has any — of which there are many doubts. My passions are unused to restraint & she is so warm — so passionate & withal so yielding in her disposition that I see no way of escape, without committing the unpardonable sin against love & gallantry. It is not in my nature to thwart the inclinations of melting maids. I retired from Church to day. Mother absences. I wrote to my father this week.

Done on Sunday 24th Oct 1841.

The workflow used to create the presentation of the Dusenbery journal begins with a pre-processing step, during which the dark backgrounds against which the pages were photographed are removed from the images. This can be done in any image editing program simply by selecting the text area of the image, inverting the selection, and filling the new selection with white. This prevents interference of non-textual marginal marks in the line detection process. More difficult source images might require more pre-processing. The potrace program relies on converting the source image to black and white, using a cutoff value to decide which pixels become black and which white based on their brightness. For this reason, eliminating “false positives” before processing improves the results.

The modified TIFF images were then converted to a format that can be processed by potrace using ImageMagick’s convert tool. Potrace was then run over the resulting bitmaps. The program's -k parameter determines the black/white cutoff point. A command like:

```
      potrace -s -k 0.6 image.pnm
```

produces an SVG file, image.svg. Since this is a UNIX system, batch commands may be used, like:

```
      for f in $(ls *.pnm); do potrace -s -k 0.6 $f; done
```

'-k' can be adjusted to optimize the results. In addition, the '-t' parameter can be used to eliminate specks in the source image. The resulting images may then be processed with the line detection algorithm, using a command like:

```
      python line_detector.py input.svg output.svg
```

The line detection program adds `svg:rect` shapes to the SVG and simultaneously ungroups the `svg:paths` so that they may be independently selected. The resulting document can then be opened, checked, and if necessary corrected using an SVG editing program, like Inkscape.

A final step in the process involves adjusting the SVG document's coordinate system to match the dimensions of the source image. An XSLT stylesheet, with the input of width and height parameters is used to adjust the document's coordinate system and scale the shapes therein.

The Dusenbery project does not use the SVG documents themselves in its presentation. Such a use is certainly possible: the source images could be embedded in the SVG, and Javascript can be used to enable an interactive interface, changing shape colors, making shapes visible or invisible, and so on. One of the central requirements of the project, however, is broad cross-browser interoperability, and Internet Explorer will not support SVG natively until version 9. Making the page images zoomable was another project requirement, so we decided to use OpenLayers and Djatoka with the OpenLayers-Djatoka plugin (http://goo.gl/0ZC46) and to export the line rectangles as OpenLayers features. Since the SVG surrogates are XML, a simple XSLT stylesheet can be used to convert them to Javascript and the TIFF images are converted to JPEG2000. In the journal, the image is displayed on the left, and the transcription on the right. When a user hovers over a line in the transcript, the line on the image is highlighted; the image can be zoomed and panned, and the highlighting functions at any zoom level.

The text-image linking scheme in Dusenbery relies on the position of lines on a page in the transcription and in the SVG. Rectangles are assigned an @id attribute like "line1", and lines in the text have @xml:ids in the form "jld-p106-3", where the -N suffix is the line number. The viewer Javascript simply activates the highlight for the corresponding feature in the image. More comprehensive solutions are possible, but are complicated by a lack of available standards. In the end, we decided to keep the demonstrator's implementation simple.

Linking

What does "linking" mean?

1. functional linking, wherein a view presents functionality such as a mouseover on a line of text causes the line in the image to be highlighted
2. semantic linking, wherein the documents make some effort to actually describe the relationship between a segment of text in a transcription or similar text document and a region of a raster or vector image

#1 is relatively easy to achieve, but somewhat brittle. A good example is the Dusenbery journal demonstrator. It relies on Javascript to paint a rectangle on top of the associated page image when the user hovers over a line in the transcription. For each page, every line element (either a <span> or a <tr>) has an id, and a Javascript file that contains associated ids and coordinates for each line in the image, when an onmouseover event is fired by a <span> in the transcription HTML view, the code uses the variable whose name corresponds to that id, and modifies the border visibility of the OpenLayers Vector Feature it references. This all works, but relies on a number of dependencies: a browser with a Javascript engine, the OpenLayers Library, the image server application and the Javascript code OpenLayers uses to load the image, and the jQuery library, used to bind the onmouseover and onmouseout events on each <span> to the code to switch the OpenLayers feature's border on or off. Without these dependencies, the linking would not function. The semantics of this linking method are quite weak—they essentially rest on the fact that the line-containing HTML elements and the variables containing the OpenLayers Vector Features share a name/id.

#2 is harder. We have problems on both sides, image and text. For the source image, we don't begin too badly: we have a raster image, with an SVG derivative, which can link to or embed the source image, using it as a background layer, for example. We have vector paths tracing the text, and rectangles denoting lines of text. The semantics of SVG are purely geometric, however. There is no notion other than overlap to signal any containment relationship between line-rectangle and paths, so the "lines" do not in fact group together or bound the traced text in any way. Adding metadata to the SVG, in the form of RDF triples, could address this problem, but here we are faced with adapting or inventing an ontology to describe the relationships. Some preliminary work was done during the grant period on this, but more is needed. On the text side, we have been assuming a TEI document. Obviously there are a variety of ways to present a text document, even using plain text, but TEI XML seems likely to provide us with sufficient semantics and hooks to achieve our goals.

Perhaps it is best to begin with a note on XML and pointing semantics. XML documents are trees. There is a root element that contains all other elements, and elements and other nodes (such as text) must nest hierarchically. There are a number of ways to reference parts of an XML document that are W3C standards, and TEI defines some additional ones, but the most well-supported of these are fragment identifiers and XPath. A fragment identifier relies on the fact that any XML element may have an id or xml:id attribute. An element with this kind of id attribute can be addressed, conveniently, with the URL of the document plus a hash '#' followed by the identifier. This means, for example, that internal links can be made just using the form ‘#id’. XPath is far more complex, using a path notation similar to directory paths on a filesystem or in a URL. With it, any node (including text nodes and attributes) in the XML tree is addressable.

The TEI Guidelines define a number of pointer schemes (see http://goo.gl/b8M3n) as extensions to the basic URI scheme, including one using XPath version 1.0, through which arbitrary segments of the document (including segments that are not compatible with XML's tree structure) may be addressed. Crucially, however, these have never been implemented, and it is not clear even what implementation would mean.

In our demonstrator source text, the Dusenbery Journal, there are three ways in which a line of text may be indicated: a <lb/> (line-beginning) tag, a <line>, containing a line of verse, and a <row> in a table. The second and third of these contain the text of the line, but the first is a self-closing tag. The semantics of TEI and XML are such that it seems safe to say that a <line> is a line of text, but a line beginning is not a line of text, only the beginning of one. We therefore have a problem similar to the one we saw in the SVG: it can be a bit hard to point to an actual line of text. And we have a concrete use case for doing so: when we generate the HTML view of the document (as in the demonstrator), we will want to surround each line with an HTML <span>, to which we can bind events triggering behaviors in the associated image. Handling this pointing function would mean taking one of at least three different approaches:

1. we could make the assertion that an <lb/> is a surrogate for a line, and that pointing to the <lb/> is sufficient. This has the advantage of simplicity, as the <lb/> can be referred to with a fragment identifier, but it does not allow us any way to easily retrieve the text of a line. Complex XSLT must be used. (NOTE: this is the approach we used, in fact, and it requires a pre-processing step on the XML, plus some rather awkward XSLT to achieve, the problem being that the tree structure of the result document is dissimilar from that of the source).
2. we could add markup to the TEI document to contain each line, using a <seg> perhaps. This has the advantage of being relatively easy to implement, and allowing us to refer to the line using a fragment identifier, and gives us the ability to retrieve a line of text, but it adds an aspect to the document's hierarchy that may be incompatible with other features of the text that we want to mark up. So we have more potential for overlap, and more mess.
3. we can try to employ TEI's pointer schemes, perhaps range() or string-range() to indicate lines. This would allow us to unambiguously indicate lines beginning with <lb/>s, but it suffers from difficulties in the creation of the pointers and from the lack of implementations. An experimental implementation of string-range() using XSLT 2.0 has been developed by the author and a colleague (http://goo.gl/pBgZ3), but it is not robust, nor standard enough for use in a production environment.

If we can effectively point to lines in the image and lines in the text, then TEI's facsimile module is adequate to the task of defining the links between text and image. As we noted above, the lack of an ability to define a coordinate system in facsimile is more than made up for by SVG's abilities in that regard. We have no need therefore to employ the coordinate-bearing attributes on <surface> or <zone>, we can simply point to the SVG and elements therein.

TEI’s facsimile module did not seem well suited to links employing an intermediate object. It is designed to link a text to an image, using pixel measurements to define the surface and zones where linked objects occur. TEI does not provide any way to define the coordinate system used in linking to an image. An independent coordinate system would enable links that could (for example) reference the physical dimensions of the source text itself, rather than surrogate images. Multiple surrogates could be mapped into that coordinate system, meaning that text-image links could encompass multiple images and would be extensible, allowing new images to be linked without the need to change the surface or zone definitions. It should be noted that modifying the SVG's coordinate system to use physical rather than digital measurements is trivial, and that placing multiple raster images into that coordinate system is fully supported.

Open Annotation

The Open Annotation Collaboration draft specification (http://goo.gl/NDgb2) provides another potential way to link image and text. OAC has a concept of "Constrained Targets" (http://goo.gl/8ZTUP), in which the SVG shapes might serve to constrain links between text and image. The OAC draft envisions an RDF representation of annotation links, which, since it relies on URIs, would be subject to the same issues discussed above. OAC might work well in concert with the SVG tracings, as it could provide semantics (e.g. "this shape contains this line of text") which are lacking in the SVG format.

Future Work

We consider this initial exploration of methods for using SVG as an intermediary in linking between TEI documents and manuscript images to be promising. The construction of a toolkit that can be used to build image processing workflows has begun, and has been successfully demonstrated in the Dusenbery Journal. The components of that toolkit are a combination of third-party Open Source software, Open Standards, and Open Source software developed for the project and hosted on GitHub at https://github.com/hcayless/img2xml.

After a prolonged infancy, the widespread adoption of SVG as a format for vector graphics on the web seems to have gained momentum. IE9 will support it, meaning that for the first time all the major web browsers will display SVG without the need for a plugin. SVG seems like a natural solution to the problem of annotating images in a generalized way, and since SVG documents can be easily edited with tools like Inkscape, injecting people into the workflow for quality assurance and editing is likely to be straightforward.

The immediate future for *img2xml* is likely to center on a) further defining and improving the standards for encoding text/image links and b) exploring the integration of the code and methods developed for the project into the Text-Image Linking Environment as a plugin.